(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
16 January 2003 (16.01.2003)

PCT

(10) International Publication Number
WO 03/005248 A2

(51) International Patent Classification[7]: G06F 17/50

(21) International Application Number: PCT/EP02/06065

(22) International Filing Date: 3 June 2002 (03.06.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0116314.6 5 July 2001 (05.07.2001) GB

(71) Applicant *(for all designated States except US)*: **MOTOROLA INC** [US/US]; 1303 E.Algonquin Road, Schaumburg, IL 60196 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **GRAS, Jean-Jacques** [GB/GB]; 4 Harding Road, Abingdon, Oxfordshire OX14 1SF (GB).

(74) Agent: **TRELEVEN, Colin**; Motorola European Intellectual Property, Midpoint, Alencon Link, Basingstoke, Hampshire RG21 7PL (GB).

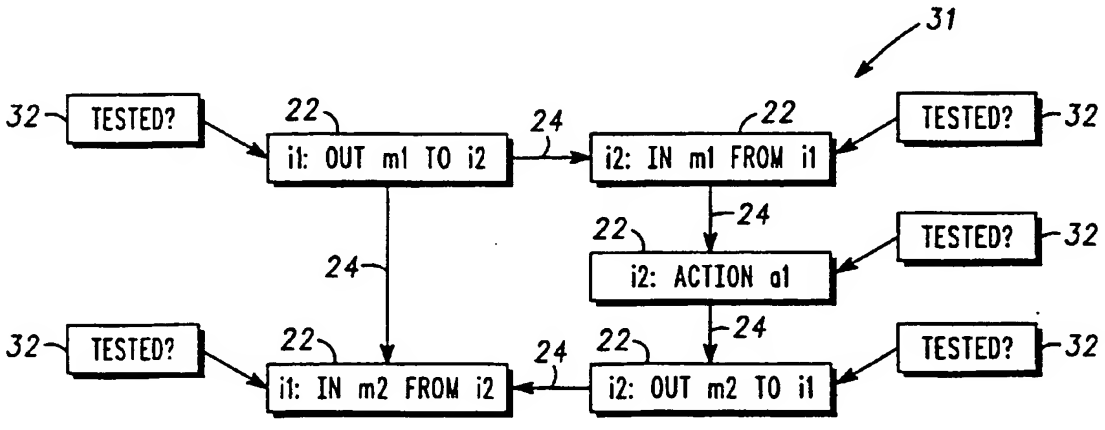
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: USING A BAYESIAN BELIEF NETWORK



(57) Abstract: A method of determining probabilities associated with a sequence of events in a system, for example an item of software or a software system being developed. A scenario, e.g. a message sequence chart, MSC (1,51), is parsed into a directed acyclic graph, DAG (21), comprising nodes (22) and edges (24), by associating the edges to messages (8,10) and sequential relations between consecutive events in the MSC (1,51). A Bayesian belief network, BBN (31), is formed using the DAG (21). BBN conditional probability tables associated with each event are computed, and the overall probabilities associated with the sequence of events is determined from the BBN conditional probaility tables. This can be used for example to provide analysis of reliability early in a soft ware development cycle. Alternative applications, e.g. to project management, are possible.

WO 03/005248 A2

Using A Bayesian Belief Network

Field of the Invention

This invention relates to the fields of statistical modelling, software engineering and reliability prediction.

Background of the Invention

Systems and software are built as a collection of components interacting according to a design specification that can be formalised using execution scenarios.

In this specification, the term "system" means any system or arrangement, physical or abstract, which can be considered to be ordered and involve performance of actions or events. As such, the term encompasses software and software systems, as well as other areas such as project plans, project management tasks, etc.

Considering that some attributes of the components are non-deterministic and known with a degree of uncertainty, the difficulty in designing an optimal system is to understand and quantify how this affects the overall software or system.

In the domain of software reliability engineering, the non-deterministic attribute would be the probability of failure of a component to execute its expected behaviour. The problem would be to evaluate the probability of failure of the overall behaviour of the system or software. These behaviours are described early in the development cycle, often using scenarios, for instance message sequence charts (MSCs) as specified by ITU Z120 standard, or UML "sequence diagrams".

If an early reliability analysis of the various specified behaviours could be performed it would then be a major factor in the design and optimisation of software testing. This is because such an analysis would give a way to develop and select the test cases in priority, to address the most error prone services.

Due to the complexity of the task in the presence of uncertainty and the lack of tools, the problem is, in practice, analysed in an empirical and intuitive way. For example, in software testing the selection of test cases is done manually by the test manager using his best judgement based on his knowledge of the weaknesses of the various components.

Some work has been published in the domain of software reliability engineering. One technique is described in a paper recently presented at ISSRE'99 by Sherif M. Yacoub et al. The paper, entitled "Scenario-based Reliability Analysis of Component-based Software", proposes to use high-level design specification (UML or message sequence charts (MSC's)) and execution scenarios to build a reliability prediction model, named "Component Dependency Graph" (CDG).

Another approach, proposed by Norman Fenton at City University, London, recommends predicting software reliability of a component based on Bayesian Belief Networks (BBN) using development process evidence.

However, this approach has the disadvantage that it neglects uncertainty representation in structural and dynamic description of interactions between software components.

The first approach above, i.e. using high-level design specification (UML or MSCs) and execution scenarios to build a reliability model, named "Component Dependency Graph" (CDG) fails to take into account uncertainty in the type of failure or reliability estimates. The second approach, recommending the prediction of software reliability of a component based on Bayesian Belief Networks (BBN), fails to take into account the design of the software.

Thus there exists a need in the field of the present invention to provide an early reliability analysis of software and system designs, wherein the abovementioned disadvantages may be alleviated.

Known prior art documents include EP-A-1,109,101, EP-A-1,065,578, WO-A-00/41123 and US-A-6,076,083.

Statement of Invention

In summary, the inventors have recognised that uncertainty in the type of failure or reliability estimates in software and system design can be intrinsically managed by employing a BBN statistical model approach. Moreover, the inference capability of BBN's makes them a choice for computing the effect of the observation of the state of variables (the "evidence") on the whole set of variables in the network. The revised probability distributions associated with each of them can help identify the most probable cause and the consequences of an observed state of a variable. Hence the present invention uses BBN's to determine probabilities associated with sequences of events.

In a first aspect, the present invention provides a method of determining probabilities associated with a sequence of

events in a system, as claimed in claim 1. In a second aspect, the present invention provides a method of evaluating the reliability of a system, as claimed in claim 11. In a third aspect, the present invention provides a storage medium storing processor-implementable instructions, as claimed in claim 16. In a fourth aspect, the present invention provides an apparatus adapted to carry out the method of the first and second aspects, as claimed in claim 17. Other aspects are as claimed in the dependent claims.

In a further aspect, the prediction of software reliability of a component based on Bayesian Belief Networks (BBN) is used to provide failure probability inputs to the BBN generated from the design of software in order to build a complete test management tool.

In addition, aspects of the present invention provide, or allow for the provision of, any one or any combination of the following:

(i) reusing scenario specifications such as Message Sequence Charts (MSC's) or UML sequence diagrams developed for high level design;

(ii) automatically translating scenarios into a Bayesian Belief Network (BBN) that offers a rigorous mathematical framework for reasoning and decision-making under conditions of uncertainty;

(iii) constructing a BBN compatible with now existing BBN tools that can deal with a high number of variables; and

(iv) allowing the integration of the resulting BBN with other BBN's that could describe other aspects.

Brief Description of the Drawings

Embodiments of the present invention will now be described, by way of example only, with reference to the accompanying drawings, in which:

FIG. 1 shows a scenario as a message sequence chart (MSC);

FIG. 2 shows the MSC of FIG. 1 represented as a directed acyclic graph (DAG);

FIG. 3 shows a Bayesian belief network (BBN) formed from the MSC of FIG. 1 and DAG of FIG. 2;

FIG. 4 is a flowchart showing process steps performed in an embodiment of the present invention;

FIG. 5 shows a further MSC;

FIG. 6 shows a computer screen obtained by executing the MSC of FIG. 5; and

FIG. 7 shows a further computer screen obtained by executing the MSC of FIG. 5.

Description of Preferred Embodiments

Reusing Message Sequence Charts

The system and software community uses Message Sequence Charts (MSC's) to represent the internal behaviour of a software application or of a system. The MSC describes the concurrent activities and relations between architectural components represented as sequences of events (also known as actions) and message passing between instances. This effort, initially triggered by the need for an accurate description of a software or system behaviour, is even more useful nowadays with the introduction of automatic test case generation tools from MSCs and in the future with component based software development proposed by the "productline" approach.

These MSCs representing dynamic relations between components in a software or system can be reused for other types of analysis that they were not primarily intended for. For instance, a system engineer could want to analyse the probability of failure of an end-to-end service for a particular scenario described by an MSC.

The message passed between instances can be viewed as expressing dependencies between components or as carrying influence from a component onto others. For instance, a sender of a message can influence the capability for a receiver to interpret the content of the message, if it does not build it according to an agreed syntax before transmitting it.

Most often this influence is non-deterministic as seen from the macro level of the component. The fact that the sender did not build the message properly may be a result of a design fault that can only be observed under certain conditions and that then occurs quasi randomly with a certain probability. The receiver may, as a result, fail to decode the message, but may also not make use of the wrong data and still proceed correctly with the reception. It may also fail to decode a correctly built message because of an intrinsic fault in its own design.

In an MSC, the transmission and the reception of a message and the actions executed as a result are called events. Events are executed sequentially within a given instance. The analysis of influence between events described previously could also be applied to two consecutive events in an instance. The type of influence would be different from a corrupted message but could relate for instance to corrupted

data shared by software modules processing the events within a particular instance.

In order to evaluate the probability of failure of the scenario, the system engineer could start applying this type of analysis from the first event initiating the service and continue until the last event that concludes the execution of the service. Numerous scenarios are usually necessary to cover the various cases of the execution of a complete service. The combination of results from individual analysis is also useful to get a realistic evaluation of the service failure probability that would be observed on the field.

The number of instances and messages in MSCs describing real systems makes this type of analysis rapidly too complex, and requires automation through the proper description of a method and its implementation with a tool.

BBN's are a technique based on Bayes theorem and graph theory to represent and compute causal influence between variables in the presence of uncertainty. A graph describes the relations between variables, and uncertainty is expressed through conditional probability tables associated with each node.

BBN's not only compute a static probability distribution associated with variables but, thanks to Bayes theorem, they can also make use of an actual observation or evidence of the state of a variable or evidence by propagating it through the network. BBN's can be used in many domains. Nowadays BBN tools bring automatic computation to the most complex BBN's.

The overall approach is the following:

(i)parse the MSC into a directed acyclic graph by associating nodes to events and directed edges to messages and sequential relations between consecutive events in the MSC. That graph constitutes the structure of the BBN and the creation of the nodes and edges can be done through the API of a standard BBN tool;

(ii) additional nodes can be created at that stage for expressing domain specific relations;

(iii) based on the supplementary information and domain specific formula, build the BBN conditional probability tables;

(iv) use the BBN propagation capability to compute the probability tables associated with each event in the MSC. The probability table associated with the last event in the MSC represents the overall result; and

(v) if necessary, evidence of a node being in a particular state can be entered, and through BBN propagation all probability tables are recomputed. The revised probabilities associated with each node indicate the most probable cause of the observed node state.

The additional domain specific information may be provided independently through the tool user interface or may be attached to the MSC, instances or messages as embedded in comments with specific identifiers. Therefore, the effort required to develop the MSCs for the initial purpose of describing the dynamic behaviour of a system could be leveraged for various probabilistic reasoning purposes.

The type of influence can be of multiple kinds, depending on the problem considered. For instance, in most cases messages passed between instances in an MSC carry data over to the

receiving instance, where these data may be necessary for the proper execution of further actions in the receiving instance. Hence, the type of influence MSCs have been primarily intending to describe is of a functional nature. However, the same MSC representation may be used with additional semantics and for example associate a random variable with each event. The messages passed between instances would then represent relations between these variables. The sequence of events found along a single instance would also imply relations between the associated variables.

This concept can be applied to a number of scenarios, as highlighted below.

Given a set of MSCs, the analysis described previously can be used to sort the set according to the probability of failure, evaluated for each MSC, and use this result to prioritise the testing of the software or system.

Since the MSC and the associated random variables define the influence between components, one could also try to use an observation of the state of a particular node in an instance to infer the most probable cause of the situation observed. In other words, this could be used to perform a diagnostic of a situation observed, like characterising a test case failure to determine the faulty component.

Alternatively, in a different domain of interest but using the same MSC that describes the software or system to analyse, the processor load or any performance related attribute impacted by the reception of the message could be considered in a performance analysis.

Ignoring their sequential possibilities, MSCs could be used as well to represent the architecture of the software or system through static description of the relations between its components. Attributes for the frequency of use of each interface could be added to component reliability. Note that this "architectural MSC" could be built by combining a collection of MSCs describing all possible scenarios associated with the frequency of occurrence of each scenario (operational profile).

There is no reason why MSCs could not be used in different domains unrelated to protocols or software but where one can apply the notions of interactions between entities leading to sequences of events. The same type of analysis can then be used to evaluate the result of the relations between the random variables mapped to the events.

As an example, this could be used in a project management tool. An MSC is derived from a PERT chart and used to represent the concurrent work of a number of resources where messages represent the dependencies between tasks. Random variables represent the duration of the tasks and a dependency between tasks defines the influence between variables in the BBN model. The uncertainty of the duration of a task is expressed as a distribution and the output is the distribution of the random variable "end date".

In accordance with a first embodiment of the present invention, the use of a Bayesian Belief Network (BBN) for the evaluation of the reliability of a software system described with a Message Sequence Chart (MSC) is described. Conventional models used for estimating the reliability of a

piece of software or software systems were based on external observation of failure rate versus testing time.

This embodiment exploits inter alia the idea that the reliability of a software application is better evaluated by taking into account its internal design and components interactions. These can be described by Message Sequence Charts (MSCs) specifying the interactions between the software processes or system components. The MSC is converted into a graph that is used as the structure of the BBN. Failure probability for each component obtained from field data or other estimates can be then added to complete it.

The BBN can be used to evaluate various aspects of failure probabilities, for instance:

(i) The end-to-end failure probability for a service described by the MSC.

(ii) The failure probability at any stage of the MSC.

(iii) The impact of a change in the failure probability of an instance.

(iv) The impact of a change in the failure probability of a particular event or action.

(v) The impact of an observed failure on the failure probability at any stage of the MSC (Bayesian inference).

Construction of the BBN

A Message Sequence Chart (MSC) is a representation of the behaviour of a system, and consists of a number of parallel processes called instances that communicate by asynchronously exchanging point-to-point messages. MSCs are commonly used to describe system or software design and behaviour.

An MSC describes a particular scenario, for example the internal behaviour of a system when a particular service is successfully completed. Applied to software reliability evaluation, the MSC can represent a typical scenario representative of the internal operations in a certain situation. The instances of the MSC represent system components and the messages passed between instances, interactions between the components' related actions. The case maybe unique or can be the generic representation of interactions occurring in a class of services.

The concept could be extended for instance by combining MSCs (High-level MSCs) and making use of other features like expressions to deal with optional behaviours having no impact on service completion. This embodiment considers a single MSC with only the basic features i.e. send and receive messages and execution of internal actions.

A BBN is derived from the description of a software system design expressed through a Message Sequence Chart.

Prior knowledge of the failure probability of each system component is assumed to be obtained, for instance from previous test results or similar product history. It may also come from another BBN describing the development process and making use of process evidence to compute a prediction of failure probability of the resulting product.

The BBN features can then be used to infer the overall probability of the service to fail at various stages in the MSC. Evidence from the testing process can also be used to update the BBN and refine the predictions using an expression giving the impact of testing on failure reduction that may be

derived from previous experience. By adding information about testing done on software module(s) corresponding to each action, the improvement of the service reliability brought by testing can be evaluated.

A basic MSC can be represented as a directed acyclic graph (DAG), where vertices are events and actions encountered in each instance, and edges are the relations between them i.e. sequential within the instance and through messages between instances. Such a DAG graph can form the structure of a Bayesian Belief Network.

The principle of the algorithm is the following: the nodes of the Bayesian network are to be created from the local events encountered by an instance. More precisely, the events taken in consideration for the moment are of three different kinds:
1. an internal action, in the MSC meaning of the term (InstanceX : action actionY),
2. the emission of a message (InstanceX : out messageZ to InstanceY), or
3. the reception of a message (InstanceX: in messageZ from InstanceY).

The nodes will be linked along each instance from top to bottom. Furthermore, for each emitting node, there should be a receiving node, and in this case, there will be a link from the emitting node to the receiving node. Finally, each of the created nodes will be given one more individual parent, to indicate the testing level of that node.

FIG. 1 shows an MSC 1 as determined for this embodiment, comprising instances 2, 4, an action 6 and messages 8, 10.

FIG. 2 shows a DAG 21 formed from this MSC, and comprising nodes 22 and edges 24.

FIG. 3 shows the BBN 31 formed from this DAG, and comprising the nodes 22, edges 24. BBN 31 also shows further created nodes 32, formed as parents to the nodes 22 to indicate the testing level of the respective nodes 22, as described above.

Conditional probability tables are derived from the user input giving the residual error probability for each node or for each instance that corresponds to the case where the parent nodes are OK and the current node fails. The other states are filled automatically with the complement to one. The discretization of the state values into more than two steps is also implemented, but may be modified to fit better the particular domain. In this embodiment the values are spread in a linear fashion from the residual error to make up a sum of one.

Thus, a process has been described for determining probabilities associated with a sequence of events in a system, using BBN's. This process can be summarised in terms of process steps shown in a flowchart in FIG. 4, the process comprising:

determining a message sequence chart, MSC (step s2); parsing the MSC into a directed acyclic graph, DAG (step s4); building Bayesian belief network, BBN, conditional probability tables (step s6); computing the BBN conditional probability tables associated with each event (step s8); and determining the probabilities of the various outcomes associated with the sequence of events from the BBN conditional probability tables (step s10).

The step of determining the probabilities of the various outcomes associated with the sequence of events from the BBN conditional probability tables can be further understood from the following details.

The events in the MSC are related to an activity in the corresponding instance (e.g. instance B receives a message and then it has to process the event by decoding the message and interpreting it). Associated with this activity we can define a random variable which can take several states (e.g. "pass"-"fail", or "0%, 10%, 20%,...,100% processor load", or for a project management tool "0,1,...,9,10 hours work").

Thus each node of the BBN represents such a variable, and the BBN tool computes the probability that each variable is in a particular state (e.g. 0.03 probability that this variable is in the "fail" state and 0.97 (complement to one) probability that it is in the "pass" state.)

By looking at the node that corresponds to the end of the scenario (e.g. a telephone rings after a call has been initiated to its number) we can determine the probabilities associated with each of its possible states and therefore the outcome of the scenario. Note that we can look at any stage of the scenario by looking at the corresponding node of the BBN.

When we actually observe that the system is in a particular state at a stage in the scenario we can enter that evidence in the corresponding BBN node (e.g. "fail" =1, "pass"=0 probability) and the tool will propagate this information across the whole BBN and revise the probabilities for all node states. Then we can determine the most probable cause of

the situation by looking at all upstream nodes. We can also determine the revised probabilities for the scenario by looking at the last node.

In a project management tool, evidence that a task is finished would be entered by forcing the state corresponding to the actual duration of the task and then immediately seeing the revised distribution of the possible end date for the project.

To further clarify the benefits and results to be obtained from employing the inventive concepts of the present invention, details of a further embodiment, and results from its implementation will now be described. Unless stated or inherent, however, details are the same as in the first embodiment.

General aspects:

(i) the application is able to take a basic MSC as an input;
(ii) the MSC is transformed into a BBN;
(iii) the BBN's probability tables shall be created both from user input and reading from the database of test management software for gathering test coverage input; and
(iv) the results are displayed to the user in a user-friendly GUI. Advantageously, the software is written in Java™.

MSC requirements

For the purpose of the demonstration, the MSC language is restricted to a given subset:

(a) MSC is composed of instances, declared (or not) by an INSTANCEHEAD keyword.

(b) MSC actions are restricted to outing messages (without loss), receiving messages (without discovery), and performing actions.

(c) According to the MSC grammar, a free field, delimited by brackets, and found after message or action declaration, is available for anyone to use. A syntax for this field shall be defined so that it can be used for providing the input required from the user (not implemented yet).

(d) Any other piece of the official syntax of the MSCs is to be forbidden.

Bayesian tool requirements:

The chosen tool is Netica™, developed by Norsys Software Corp™, which fulfils all the requirements. Indeed, the Bayesian software is:

(i) able to build BBN's, and lead simple inference calculus on it;

(ii) able to learn data from a database, which shall be exploited later to fill the network's probability tables without any intervention from the user;

(iii) available as an API, so that it can be integrated into the application.

In this embodiment an MSC 51 is as shown in FIG. 5, comprising instances 52, 54, actions 56, 58, and message 59.

When implemented, this embodiment provides the screens shown in Fig. 6 and FIG. 7. FIG. 6 shows a screen 61 in which the MSC structure 62 appears in the left frame 64, and individual components 66 can be accessed for setting values (residual error probability or testing status) and for viewing the derived probabilities expressed as bar graphs 68 at the bottom of each window.

The BBN can also be saved and viewed using Netica's™ graphical user interface (GUI). The same probability results as displayed by the Netica™ GUI is shown on screen 71 in FIG. 7.

Interfacing with a test management system

A test management system that keeps records of test cases and test results using an Oracle™ database is used.

A query mechanism may be built within the above embodiments to connect and send a SQL query to the TMS database through a Java™ JDBC interface. In the TMS database a particular test suite is designed for the testing of the MSC under analysis. It contains a number of test cases for each send or receive event and actions in the MSC, each corresponding to a node in the BBN. For each test case the test results (EXECs in TMS terminology) can be retrieved by an SQL query and used to identify the testing status of each node in the BBN. The testing status can then be automatically updated in the BBN and a new reliability estimate produced by inference. A TMS database screen with the test suite corresponding to the MSC and the various EXECs reported can be provided.

In the above embodiments, MSC's were employed as the scenario representation. It will be appreciated, however, that the present invention may be applied to any suitable formal scenario specification that represents relations between entities.

It will be understood that the implementation of the embodiments described above offer the possibility of the following advantages either singly or in any combination:

(i) introducing the notion of uncertainty for all variables contributing to the probability of failure of the software components;

(ii) making use of BBN technology to perform the uncertainty calculations;

(iii) being able to automatically process the input scenarios and convert them into the predictive model;

(iv) the concepts being applicable to domains other than software engineering; and

(v) offering reuse of system design methodology using message sequence charts (MSC) from requirements and design phases.

The above embodiments can be implemented by configuring or adapting any suitable apparatus, for example a computer or other processing apparatus. Alternatively, the processes described can be implemented by a processor implementing processor-implementable instructions and/or stored on a suitable storage medium, such as computer memory, hard disk, floppy disk, ROM, PROM etc. the processor may be a computer, a network of computers, or one or more dedicated processors.

Claims

1. A method of determining probabilities associated with a sequence of events in a system, characterised by:

determining a scenario (1, 51) representing behaviour of the system, the scenario (1, 51) comprising concurrent entities (2, 4, 52, 54), messages (8, 10, 59) and events (6, 56, 58);

parsing the scenario (1, 51) into a directed acyclic graph (DAG 21), comprising nodes (22) and edges (24), by associating the nodes to the events, and by associating the edges to messages and sequential relations between consecutive events;

building a Bayesian belief network, (BBN 31), from the directed acyclic graph (DAG 21) and conditional probability tables;

computing the Bayesian belief network (BBN 31) conditional probability tables associated with each event; and

determining the probabilities associated with the sequence of events from the Bayesian belief network (BBN 31).

2. A method according to claim 1, wherein the probability provided by the BBN node associated with the last event in the scenario (1, 51) is the probability of each possible outcome of the sequence of events.

3. A method according to claim 1 or 2, wherein the step of building the DAG and the conditional probability tables is based on supplementary information and domain specific formulae.

4. A method according to claim 3, wherein the supplementary information comprises prior knowledge of variables associated with the events or concurrent entities.

5. A method according to any preceding claim, wherein, in the step of parsing the scenario (1, 51) into a DAG, the nodes and edges of the DAG are created using an application programming interface (API) of a general BBN tool.

6. A method according to any preceding claim, wherein, after the parsing step, additional nodes can be created at that stage for expressing domain specific relationships.

7. A method according to claim 6, wherein the additional nodes comprise nodes (32) indicating a testing level of the original nodes (22).

8. A method according to any preceding claim, further comprising entering evidence of a node being in a particular observed state and re-computing, through BBN propagation, all the BBN probability tables, such that the revised probabilities associated with each node indicate the most probable cause and the consequences of the particular observed node state.

9. A method according to any preceding claim, wherein the steps are applied for developing, adapting or using software, or a software system.

10. A method according to any preceding claim, wherein the steps are applied for managing a project.

11. A method of evaluating the reliability of a system, comprising using the method steps of any preceding claim, and wherein the sequence of events represents a failure mode.

12. A method according to claim 11, wherein the BBN is used to infer the overall probability of the system failing at various stages in the scenario (1, 51).

13. A method according to claim 11 or 12, wherein the BBN is used to evaluate any one or any combination of the following set of failure probabilities:
(i) the end-to-end failure probability for a service described by the scenario (1, 51);
(ii) the failure probability at any stage of the scenario (1, 51);
(iii) the impact of a change in the failure probability of an instance;
(iv) the impact of a change in the failure probability of a particular event or action; and
(v) the impact of an observed failure on the failure probability at any stage of the scenario (1, 51).

14. A method according to any of claims 11 to 13, wherein the steps are repeated for one or more further scenarios, the results from each scenario are compared and sorted according to the probability of failure evaluated for each scenario, and the resulting order is used to prioritise testing of the system.

15. A method according to any of claims 11 to 14, wherein the system is an item of software or a software system.

16. A method according to any of claims 1 to 15, wherein the scenario (1, 51) is specified using a Message Sequence Chart, MSC (1, 51), or UML sequence diagrams.

17. A storage medium storing processor-implementable instructions for controlling a processor to carry out the method of any of claims 1 to 16.

18. Apparatus adapted to carry out the method steps of any of claims 1 to 16.

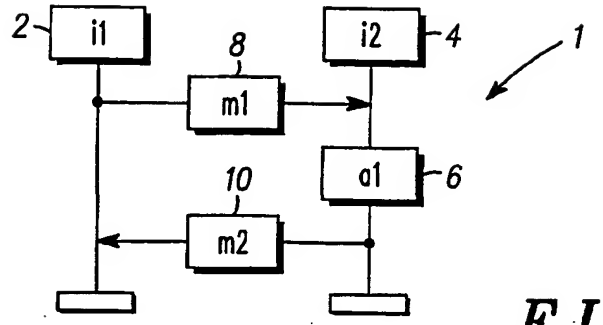


FIG. 1

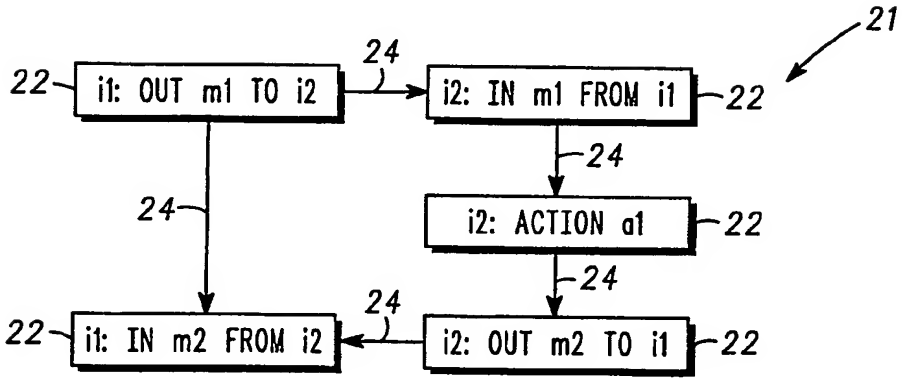


FIG. 2

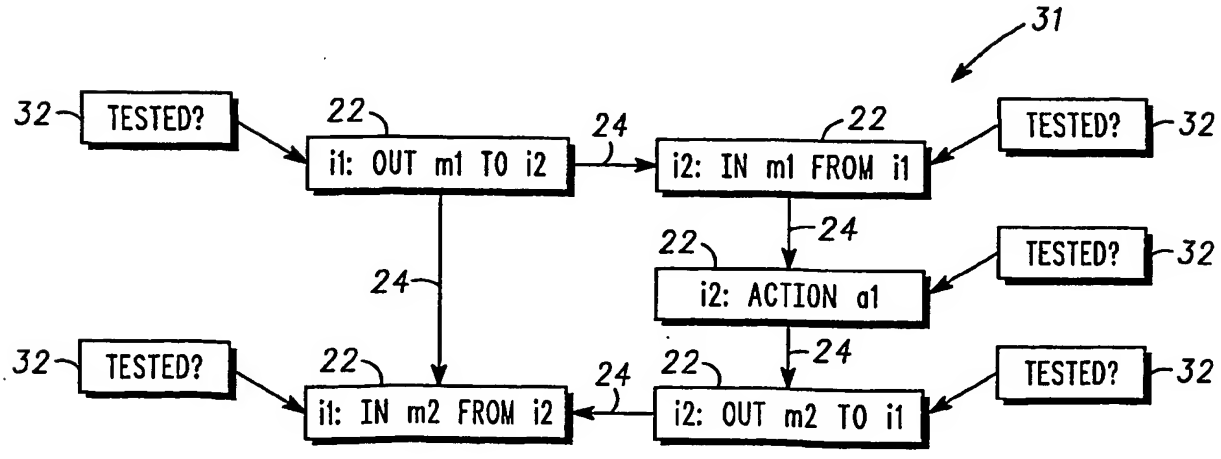


FIG. 3

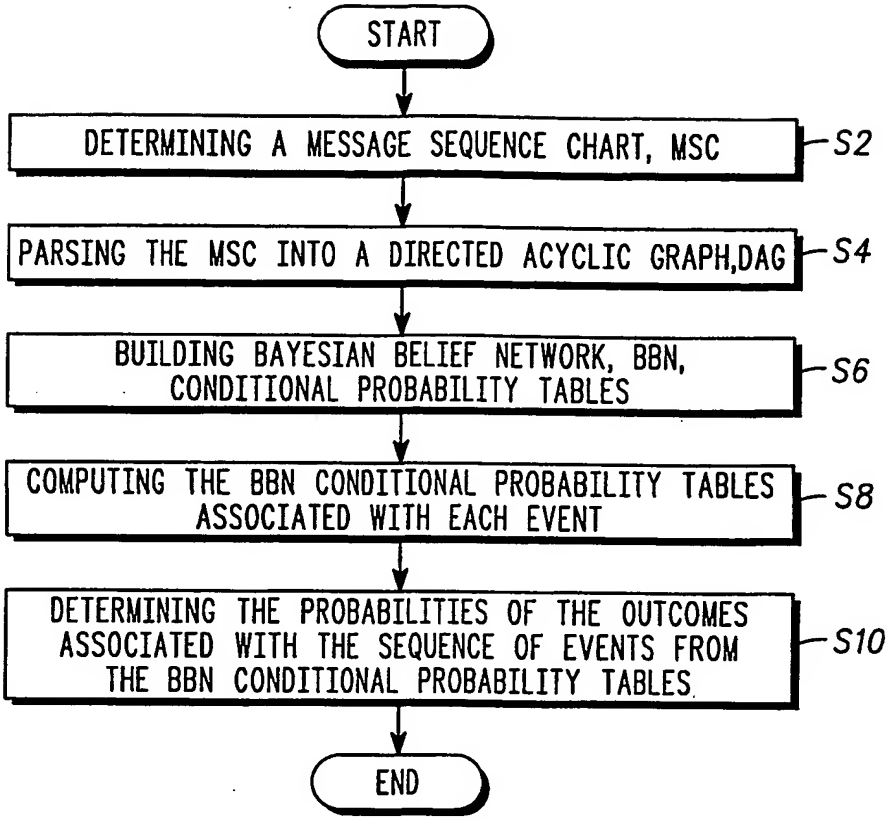

START
DETERMINING A MESSAGE SEQUENCE CHART, MSC
S2
PARSING THE MSC INTO A DIRECTED ACYCLIC GRAPH,DAG
S4
BUILDING BAYESIAN BELIEF NETWORK, BBN, CONDITIONAL PROBABILITY TABLES
S6
COMPUTING THE BBN CONDITIONAL PROBABILITY TABLES ASSOCIATED WITH EACH EVENT
S8
DETERMINING THE PROBABILITIES OF THE OUTCOMES ASSOCIATED WITH THE SEQUENCE OF EVENTS FROM THE BBN CONDITIONAL PROBABILITY TABLES
S10
END

FIG. 4

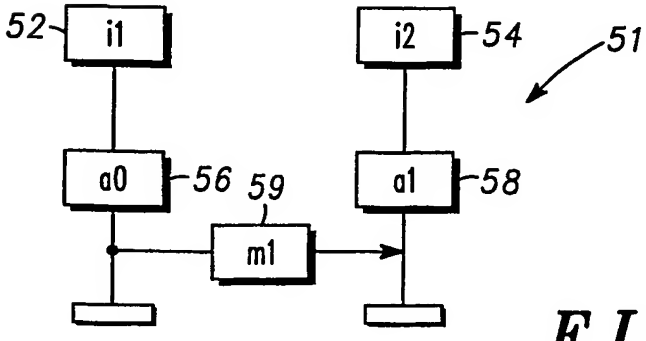

52
i1
i2
54
51
a0
56
59
a1
58
m1

FIG. 5

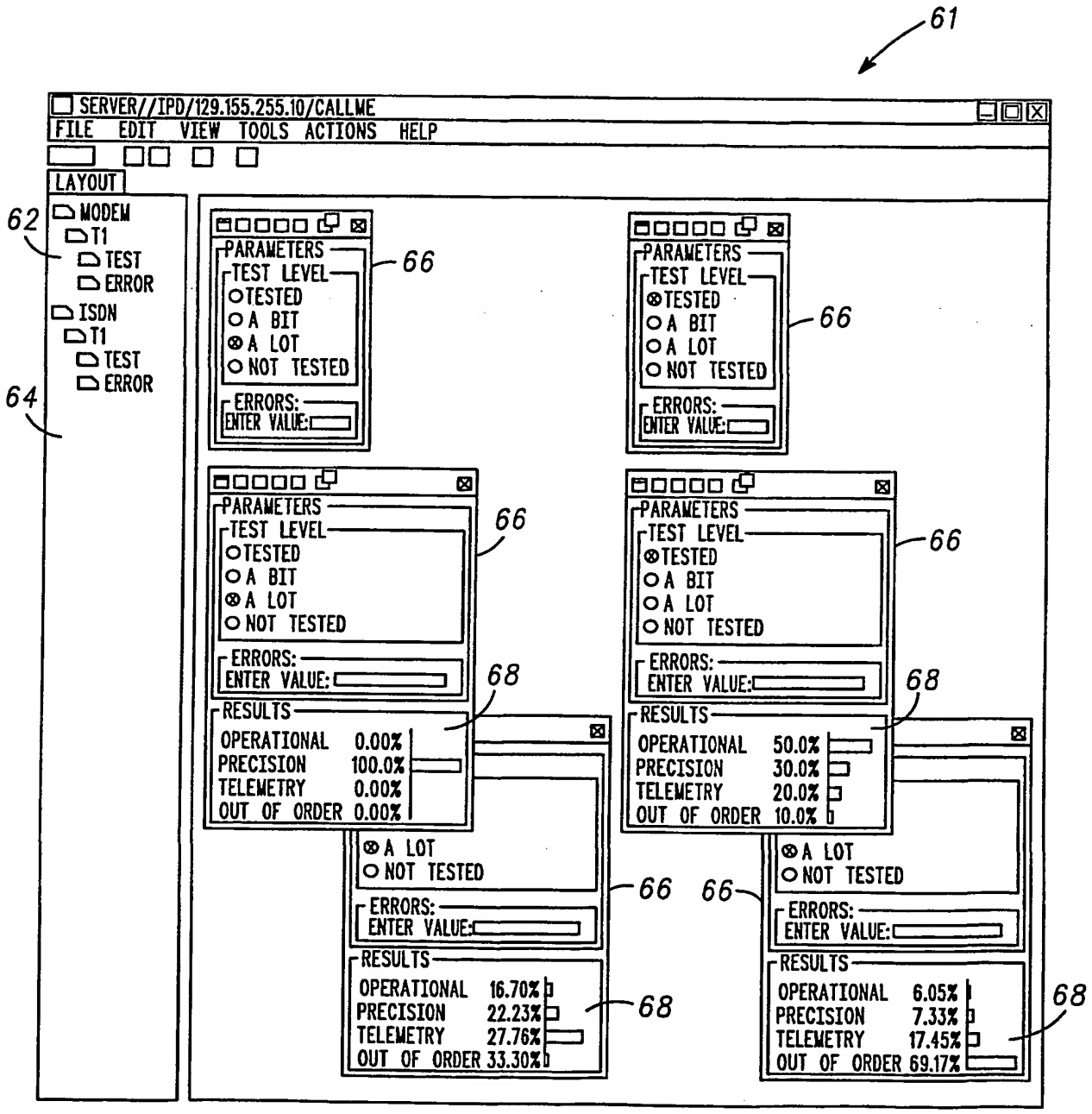

61
SERVER//IPD/129.155.255.10/CALLME
FILE EDIT VIEW TOOLS ACTIONS HELP
LAYOUT
62
MODEM
T1
TEST
ERROR
ISDN
T1
TEST
ERROR
64
PARAMETERS
TEST LEVEL
TESTED
A BIT
A LOT
NOT TESTED
ERRORS:
ENTER VALUE:
66
RESULTS
OPERATIONAL 0.00%
PRECISION 100.0%
TELEMETRY 0.00%
OUT OF ORDER 0.00%
68
OPERATIONAL 50.0%
PRECISION 30.0%
TELEMETRY 20.0%
OUT OF ORDER 10.0%
OPERATIONAL 16.70%
PRECISION 22.23%
TELEMETRY 27.76%
OUT OF ORDER 33.30%
OPERATIONAL 6.05%
PRECISION 7.33%
TELEMETRY 17.45%
OUT OF ORDER 69.17%

FIG. 6

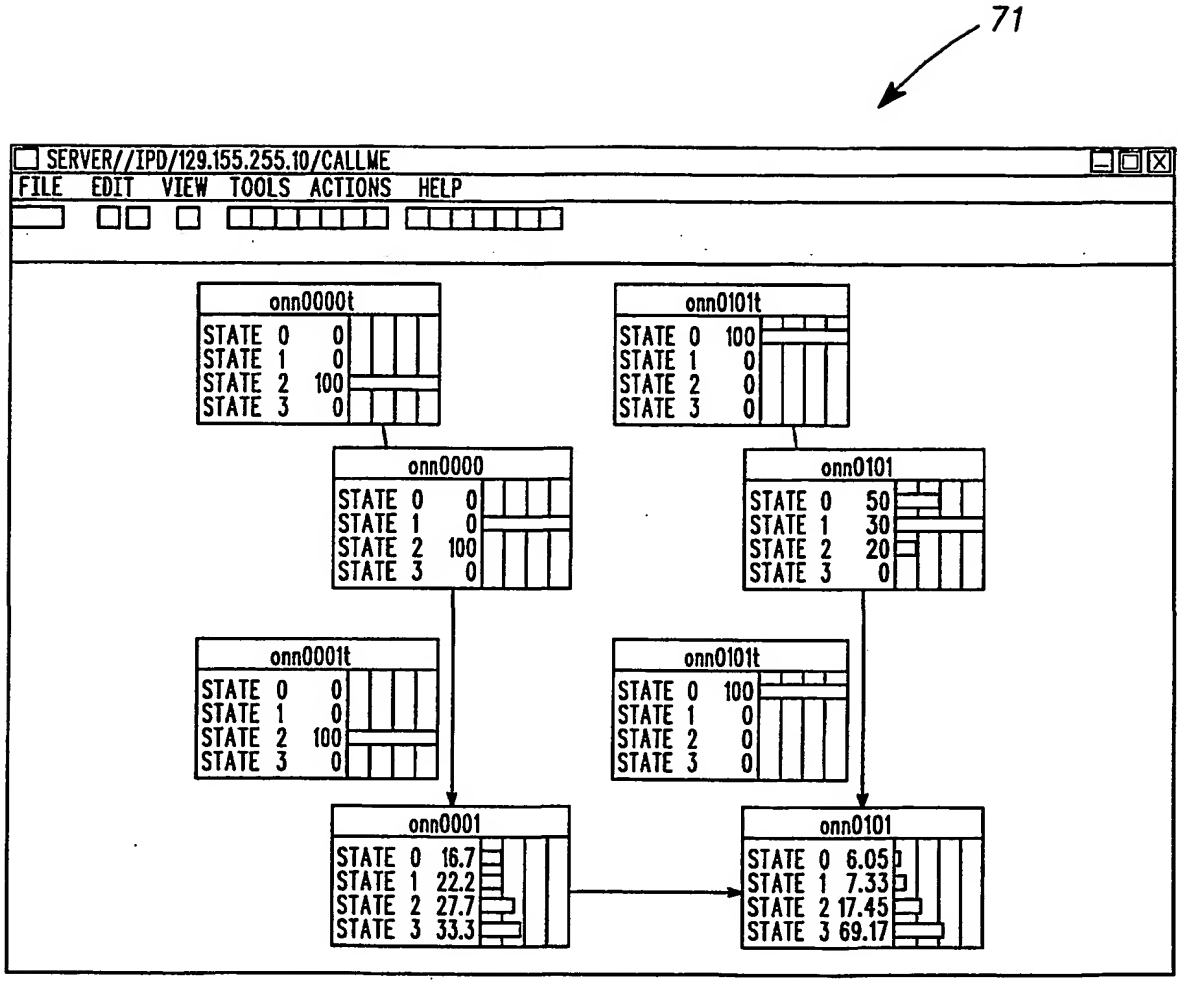

71
SERVER//IPD/129.155.255.10/CALLME
FILE EDIT VIEW TOOLS ACTIONS HELP
onn0000t
STATE 0 0
STATE 1 0
STATE 2 100
STATE 3 0
onn0101t
STATE 0 100
STATE 1 0
STATE 2 0
STATE 3 0
onn0000
STATE 0 0
STATE 1 0
STATE 2 100
STATE 3 0
onn0101
STATE 0 50
STATE 1 30
STATE 2 20
STATE 3 0
onn0001t
STATE 0 0
STATE 1 0
STATE 2 100
STATE 3 0
onn0101t
STATE 0 100
STATE 1 0
STATE 2 0
STATE 3 0
onn0001
STATE 0 16.7
STATE 1 22.2
STATE 2 27.7
STATE 3 33.3
onn0101
STATE 0 6.05
STATE 1 7.33
STATE 2 17.45
STATE 3 69.17

FIG. 7